A Hard Crystal - Implementing Dilithium on Reconfigurable Hardware

Georg Land[1,2], Pascal Sasdrich[1], and Tim Güneysu[1,2]

[1] Ruhr University Bochum, Horst Görtz Institute for IT Security, Germany
[2] DFKI GmbH, Cyber-Physical Systems, Bremen, Germany
firstname.lastname@rub.de

Keywords: FPGA · Dilithium · PQC

Abstract. CRYSTALS-Dilithium as a lattice-based digital signature scheme has been selected as a finalist in the Post-Quantum Cryptography (PQC) standardization process of NIST. As part of this selection, a variety of software implementations have been evaluated regarding their performance and memory requirements for platforms like x86 or ARM Cortex-M4. In this work, we present a first set of Field-Programmable Gate Array (FPGA) implementations for the low-end Xilinx Artix-7 platform, evaluating the peculiarities of the scheme in hardware, reflecting all available round-3 parameter sets. As a key component in our analysis, we present results for a specifically adapted Number-Theoretic Transform (NTT) core for the Dilithium cryptosystem, optimizing this component for an optimal Look-Up Table (LUT) and Flip-Flop (FF) utilization by efficient use of special purpose Digital Signal Processors (DSPs). Presenting our results, we aim to shed further light on the performance of lattice-based cryptography in low-cost and high-throughput configurations and their respective potential use-cases in practice.

1 Introduction

In the light of continuous progress and advancement on the development of quantum computers, security of existing public-key cryptographic schemes starts to crumble [11]. While most existing and currently deployed schemes rely on the hardness of *integer factorization* or computing *discrete logarithms*, broken by Shor's quantum algorithm [14], given that an attacker has access to a large-scale quantum computer, a call for the design, proposal, and standardization of new post-quantum secure schemes for Key Encapsulation Mechanism (KEM) and digital signatures has been initiated by the United States National Institute for Standards and Technology (NIST) in 2017 [9].

After two competitive rounds of thorough scrutiny and examination, NIST announced the seven finalists from the initial field of 69 candidates in 2020 which still have to undergo further evaluation in a third and final round. Moreover, the seven finalists can be categorized into the four key establishment schemes, Classic

McEliece, Kyber, NTRU, and Saber as well as the three digital signature schemes Dilithium, Falcon, and Rainbow.

Interestingly, five out of the seven remaining finalists are using hard lattice problems as fundamental security assumption. Along with Falcon [6], Dilithium [4] is one of the two remaining lattice-based digital signature schemes, while Rainbow is based on multivariate cryptography instead. Further, Dilithium and Kyber are part of the Cryptographic Suite for Algebraic Lattices (CRYSTALS) using structured lattices to allow fast arithmetic and enable compact key, ciphertext and signature sizes. More precisely, the underlying polynomial ring enables efficient polynomial multiplication leveraging the Number-Theoretic Transform (NTT).

While literature is rich in efficient and optimized implementations on lattice-based KEMs, to date, lattice-based digital signature schemes are mostly neglected. In particular, efficient implementation of lattice-based signature schemes in reconfigurable hardware urgently needs to be investigated in order to guide and support the selection of the future post-quantum cryptography standards. In this regard, we are only aware of a two existing hardware implementations of Dilithium [12,15], while several optimized software implementations, e.g., targeting AVX2 [3] or Cortex-M4 [7] architectures, have been presented recently. Further, even though the design in [12] has been implemented on a high-performance Virtex-7 Field-Programmable Gate Array (FPGA), it does not exploit important features of modern reconfigurable hardware architectures efficiently. For this, we present a novel set of efficient and compact FPGA implementations specifically targeting a low-end Xilinx Artix-7 series through evaluating the peculiarities of the Dilithium digital signature scheme for efficient and clever mapping into modern FPGA features and components[1].

Contribution. For this, our contribution can be summarized as follows:

- A specifically crafted and adapted NTT making extensive use of Digital Signal Processors (DSPs) is presented to exploit peculiarities and features of modern low-end FPGAs.
- Our Dilithium core is compact and comprehensive, providing functionalities for key generation, signature generation, signature verification, precomputation, arbitrary-length message digesting, and packing and unpacking keys and signatures.
- For Dilithium-III, our core uses 30k Look-Up Tables (LUTs), 11k Flip-Flops (FFs), 45 DSPs and 23 Block-RAMs (BRAMs) with $f_{max} = 142$ MHz. For key generation, our core is capable of performing 4290 OP/s, for signature generation 1351 OP/s and for signature verification 11751 OP/s.

Related Work. Many lattice-based schemes have been proposed in recent years and there is a wide variety of implementations in hardware. The first implementation of a lattice-based signature scheme was proposed by Güneysu et al. [8]

[1] Our implementation will be publicly available at: `https://github.com/Chair-for-Security-Engineering/dilithium-artix7`.

in 2012. Pöppelmann et al. extend this work in [10]. For Dilithium, to the best of our knowledge, there are two other implementations on FPGA: The first one uses a High Level Synthesis (HLS) approach [15] for Artix-7 and the second one using Virtex-7 as platform [12]. Other post-quantum secure signature schemes that have been implemented in reconfigurable hardware include Rainbow [5] and SPHINCS [1]. Furthermore, efficient implementation of the NTT in hardware has been research very well. Roy et al. presented an efficient design that uses two merged NTT layers [13]. Banerhee et al. presented an Application-Specific Integrated Circuit (ASIC) design of the NTT that can be used to accelerate multiple schemes [2]. Finally, Zhang et al. present a way to integrate the post-processing of the inverse transformation into the main computation resulting in a low-complexity implementation [16].

2 Preliminaries

2.1 Notation

Throughout this work, we will use and assume the following notation. Let n and q be two integers, such that $n = 256$ and $q = 2^{23} - 2^{13} + 1$. Further, let $\mathcal{R}_q$ be a polynomial ring with $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$. In addition, let us denote vectors in bold lower-case letters, e.g., $\mathbf{v}$, while matrices are denoted in bold upper-case letters, e.g., $\mathbf{A}$. Polynomials in NTT domain are indicated by a hat.

2.2 Number-Theoretic Transform

The NTT, as used in Dilithium, can be seen as a discrete Fourier transform over polynomials in $\mathcal{R}_q$, where the complex arithmetic is replaced by the modular arithmetic of the polynomial coefficients. Since the ring structure enables positive wrapped convolution, we can use an n-point NTT for fast polynomial multiplication by transforming both factor polynomials to the NTT domain, multiplying coefficient-wise in NTT domain, and then applying the inverse transform to the result to obtain the final product polynomial.

2.3 CRYSTALS-Dilithium

In July 2020, the NIST announced the 7 finalist and 8 alternate candidates for the Post-Quantum Cryptography (PQC) standardization competition, with both schemes of the CRYSTALS being selected as finalist for their respective categories. In particular the digital signature scheme Dilithium has undergone a thorough scrutiny during the competition process and since then reached version 3.1 [4], while most recently some major changes and updates for the various security parameter sets have been presented.

In general, the Dilithium digital signature scheme has been designed to adopt simple and secure design principles, in particular substituting discrete Gaussian sampling in favor of uniform sampling. In addition, all remaining fundamental

operations have been carefully chosen such that they easily can be performed in constant time. Aiming at long-term security, the different security levels and parameters have been chosen conservatively while endeavoring to minimize the combined size of public key and signatures. Eventually, the modular construction of Dilithium favors efficient and highly optimized implementations across all security levels and parameter sets as the main operations rely on SHAKE-128 or SHAKE-256 and the multiplication in the polynomial ring $\mathcal{R}_q$, regardless of the security level. Instead, higher or lower security is only achieved through addition or reduction in the number of operations performed in $\mathcal{R}_q$.

Further, as a digital signature scheme, Dilithium provides the following three core methods for *key generation*, *signature generation*, and *signature verification*.

Key Generation. For key generation, the respective algorithm generates a $k \times l$ matrix $\mathbf{A}$ such that each entry in the matrix is a polynomial of the ring $\mathcal{R}_q$. Using randomly sampled vectors $\mathbf{s}_1$ and $\mathbf{s}_2$, with coefficients being small elements from $\mathcal{R}_q$, the second part of the public key is generated as $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$, performing all algebraic operations over $\mathcal{R}_q$.

Signature Generation. The fundamental operation of Dilithium is the generation of digital signatures. For this, the signing algorithm chooses a masking vector $\mathbf{y}$ in order to compute $\mathbf{A}\mathbf{y}$ and chooses $\mathbf{w}_1$ to be the *higher-order* bits of the coefficients in the resulting vector. The challenge c, a polynomial in $\mathcal{R}_q$ is the hash of the message and $\mathbf{w}_1$ and is used to generate the potential signature $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$. Using rejection sampling, leakage of the secret key is prevented, at the penalty of repeating the signature generation process if the signature fails the security and correctness checks.

Signature Verification. For signature verification, $\mathbf{w}_1'$ is set to be the *higher-order* bits of $\mathbf{A}\mathbf{z} - c\mathbf{t}$. A signature is valid and accepted if the coefficients in $\mathbf{z}$ pass the security check and if c is the hash of the message and $\mathbf{w}_1'$.

Parameter Sets. With introduction of version 3.1 of the Dilithium algorithm specification, the list of supported security parameter sets has been adjusted for the three NIST security levels II, III, and V. Since the operations in $\mathcal{R}_q$ do not change for the different parameter sets, the performance-critical dimensions of $\mathbf{A}$ are adjusted, resulting in an increased or reduced number of operations, depending on the targeted security level.

3 Design Considerations

Modern FPGA generations are equipped with a multitude of general purpose logic. However, for certain applications, highly optimized special purpose components such as very compact and optimized DSP cores are provided, offering

efficient and fast integer arithmetic operations, or BRAM, offering compact true dual-port memory banks for easy storage of larger amounts of data. Given this, our primary design goal was to reduce the footprint of our architecture in terms of general purpose components such as LUTs and FFs, as these components usually are limited in larger systems.

3.1 Arithmetic

As a first step, we opted to implement most coefficient-level arithmetic in DSP modules for fast and efficient arithmetic, since these components are rarely used due to their special purpose character. More precisely, we exploit several special features of modern Xilinx DSP blocks, including:

Runtime reconfiguration. During design and synthesis time, the DSP can be configured to provide different functionalities during runtime. Based on this, we configured all our instantiated DSP modules to provide multiple different arithmetic operations, allowing to re-use the same DSP for different operations, hence resulting in a highly integrated and optimized design with respect to area and utilization.

Pre-addition. Besides fast integer multiplication, each DSP unit is equipped with a pre-adder stage, allowing to merge multiple arithmetic operations within a single DSP.

Single Instruction Multiple Data. Although each DSP unit can perform up to 48-bit wide additions, we opted to use the DSP cores in a Single Instruction Multiple Data (SIMD) fashion, allowing to perform two 24-bit additions instead, perfectly fitting the constrains of underlying arithmetic operations in the polynomial ring.

Number-Theoretic Transform. Further, for the design of the NTT, we adapt the fundamental ideas from [16] and transferred the concepts to the polynomial ring and modulus of Dilithium. Analogous to their approach, our NTT architecture relies on two optimized and reconfigurable Butterfly Units (BFUs).

3.2 Memory

Besides efficient arithmetic, a sophisticated memory architecture and layout is required to store and load coefficients and polynomials efficiently during arithmetic operations. Given the design considerations for our arithmetic modules, in particular the NTT unit, we identified the following two constraints for our memory architecture:

1. Due to the NTT architecture, our design must be capable of reading and writing up to four coefficients simultaneously. For this, we decided to use four simple dual port BRAMs to store polynomials. More precisely, we use four parallel 18K BRAM instances for this, each of them holding up to

512 coefficients. This means, since for a single polynomial only 64 coefficients are stored per BRAM, we can fill the four 18K BRAM units with up to eight full polynomials.
2. The memory layout has to be adjusted such that read and write conflicts are avoided for each operation. In particular, the layout has to ensure that the coefficients of the polynomials are distributed among the BRAMs such that we always can read or write data during the arithmetic operations without stalling due to memory access conflicts. For further details, we refer to [16].

Given that we can store up to 8 polynomials using four BRAM units, the total number of BRAM instances is governed by the security level. In particular, we need to hold $k \cdot l + 3l + 6k + 1$ polynomials in total, i.e., 53 polynomials for level II, 82 polynomials for level III, and 127 polynomials for level V. Fortunately, we were able to identify efficient memory mappings for each parameter set, such that it requires only requires $\lceil 4(kl + 3l + 6k + 1)/8 \rceil$ 18K BRAM primitives and further enables the following operations in a pipelined or parallel fashion:

- During matrix-vector multiplication, the vector elements are transformed sequentially to NTT domain. Upon completion of the transformation, the multiply-accumulate module updates the resulting vector elements through coefficient-wise multiplication with the $\hat{\mathbf{A}}$ polynomials.
- In pre-computations for signature generation and verification, the matrix $\hat{\mathbf{A}}$ is expanded and in parallel, NTTs of $\mathbf{s_1}$, $\mathbf{s_2}$, $\mathbf{t_0}$ and $\mathbf{t_1}$ can be performed.
- The norm check of $\mathbf{z}$ can be performed in parallel to sampling c.

3.3 Functionality

In order to provide an integrated and self-contained core for generation and verification of digital signatures based on the Dilithium scheme, our architecture needs to support the full set of the following operations:

KeyGen	Generation of a key from a given seed.
$\mathbf{Sign_{pre}}$	Expansion of $\hat{\mathbf{A}}$ and pre-computation of $\hat{\mathbf{s}}_\mathbf{1}$, $\hat{\mathbf{s}}_\mathbf{2}$, and $\hat{\mathbf{t}}_\mathbf{0}$.
Sign	Signature computation.
$\mathbf{Verify_{pre}}$	Expansion of $\hat{\mathbf{A}}$ and pre-computation of $\hat{\mathbf{t}}_\mathbf{1}$.
Verify	Signature verification.
$\mathbf{Digest_{msg}}$	Hashing of arbitrary-length messages along with tr (of the public key).
Store	Storing and unpacking public keys, secret keys, signatures, or seeds.
Load	Packing and sending public keys, secret keys, or signatures.

4 Implementation on Reconfigurable Hardware

In this section, we outline the basic architecture of our comprehensive Dilithium architecture. In particular, our construction exploits special purpose units and

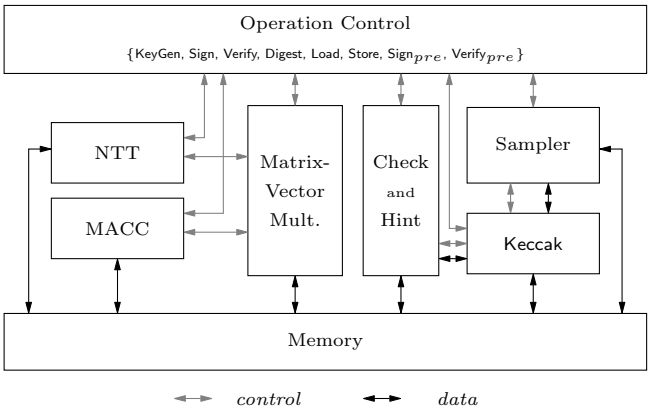

Fig. 1: Dilithium High-Level Architecture

features of a modern Xilinx 7-series Zynq System-on-Chip (SoC) platform hosting an Artix-7 FPGA (XC7Z020).

4.1 Architectural Details

The high-level architecture of our implementation is shown in Fig. 1. All basic arithmetic operations are performed by the *NTT*, *Multiply-Accumulate (MACC)*, and *Matrix-Vector Multiplication* units. However, even though the matrix-vector multiplication serves as master and control unit for the NTT and MACC cores, both sub-cores must be accessible from the global operation control unit as well to provide auxiliary support for additional arithmetic operations. Besides, the check units directly access polynomials in the memory for norm checking and provide the check result to the operation control module. The *Sampler* module controls and accesses the *Keccak* hash core in order to buffer the hash output before writing the uniformly generated random samples to memory. However, the Keccak-based hash core is also accessible from the operation control unit, mostly required for random seed expansion. Finally, the *hint* modules control read and write access to the hint registers in the memory unit. Further, as already highlighted in Section 3, the memory unit consists of several BRAMs for the intermediate polynomials, two 512-bit registers to store ρ' and μ as well as some additional 256-bit registers for ρ, $\tilde{c}$, tr, K, and the seed for the key generation.

Number-Theoretic Transform. As already mentioned in Section 3, our NTT implementation follows the design principles of [16], however, applying the following two modifications. First, we make use of the *true* dual-port capabilities of the BRAM modules, enabling our design to read two twiddle factors simultaneously in the lowest NTT layer and thus still allowing processing four coefficients at the same time. Second, the stored twiddle factors for the inverse transform

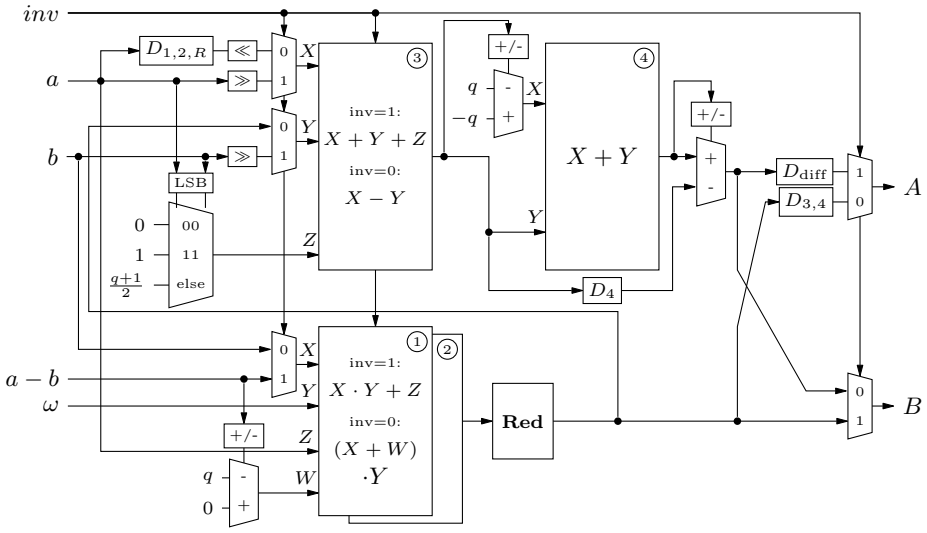

Fig. 2: Architecture of the BFU. DSPs are numbered, D_i are shift registers that compensate for the DSPs or the reduction as given in their respective index. D_{diff} compensates for the difference of cycle counts between $D_{1,2,R}$ and $D_{3,4}$

are pre-multiplied by a factor of 2^{-1} in order to avoid any additional modular division logic.

At the core of the NTT, we instantiate two independent BFUs, as depicted in Fig. 2. More precisely, each BFU receives two unsigned 23-bit coefficients, an unsigned 23-bit twiddle factor, and the *signed* 24-bit value $a-b$. Note, however, that this value can be pre-computed for both BFUs simultaneously using a single DSP in SIMD mode.

Forward Number-Theoretic Transform. In general, the forward NTT computes two values A and B such that $(A, B) := (a+b\cdot\omega, a-b\cdot\omega)$, given that ω denotes the pre-computed twiddle factor. For this, we use the DSPs 1 and 2 to compute $a+b\omega$. More precisely, we need to combine two DSPs for this operation since each DSP itself can only perform signed 25×18-bit multiplications. However, when combining DSPs for larger multiplications, we can leverage a dedicated low-latency cascade path. After multiplication, the resulting product is reduced to a representative in $[0, q)$ and already provides the first part of the forward NTT computation. Further, subtracting the first part from $2a$ and adding or subtracting q (depending on the sign of the subtraction result), we obtain the second part of the forward NTT output.

In addition, for increased throughput, the BFUs have been pipelined, using shift register instances to delay the input a of the third DSP. More specifically, the first part of the result is also delayed through a shift register in order to return both parts of the forward NTT computation simultaneously.

Inverse Number-Theoretic Transform. Similar to the forward NTT, the inverse NTT computes two values A and B, such that $(A, B) := (2^{-1}(a+b), (a-b)\omega)$. However, as already mentioned before, this time the operand ω for the inverse NTT is already pre-processed to incorporate the factor 2^{-1}.

Here, ω and the pre-computed, signed value $a-b$ are used as input for the multiplication DSPs 1 and 2. Further, depending on the sign bit of the value $a-b$, we choose between adding q or 0 using the pre-adder stage of the multiplication DSPs to obtain a positive multiplication result. Finally, the multiplication result is then reduced and serves as output. Besides, the second part of the output is designed to be $2^{-1}(a+b)$. For this, we use DSP 3 as 3-input adder with inputs $\lfloor a/2 \rfloor$, $\lfloor b/2 \rfloor$ and either 1 (if both least signification bits (LSBs) of a and b are 1), or $(q+1)/2$ (if the LSB of either a or b is 1), or 0 otherwise. Since the result of this operation might be greater or equal to q, we use the fourth DSP to subtract q from the result. The second part of the BFU output is then chosen between the output of DSPs 3 and 4.

Multiply-Accumulate. The second arithmetic core is used to perform *multiply-accumulate* operations. More specifically, this core is designed to perform four computations per clock cycle in parallel in order to make full use of the available memory bandwidth. It consists of eight DSPs and four reduction modules. Each two DSPs perform one of the following operation, while the result then is fed into the reduction module.

- $a \cdot b + c$: The first DSP performs the multiplication of a with the lower 17 bits of b and the addition. The second DSP multiplies a with the remaining upper bits of b and updates the first result to the final 46 bit value that is then fed into the reduction module.
- $a + b$: The first DSP computes the sum, while the second one subtracts q. Eventually, the result of the second DSP is selected if it is non-negative, else the result of the first DSP is selected.
- $b - a$: The first DSP computes the subtraction, while the second one adds q. Eventually, the result of the first DSP is selected as output if it is positive, else the output of the second DSP is selected.

Note that for operations without multiplication, the reduction module can be bypassed, resulting in a lower latency. Again, this module is fully pipelined, allowing to process an entire polynomial within 64 cycles (in addition to the initial pipeline length).

Matrix-Vector Multiplication. This module controls both the NTT module and the MACC module to (1) transform the polynomials in the input vector into NTT domain and (2) perform a matrix-vector multiplication with $\hat{\mathbf{A}}$. The resulting polynomial vector is then in NTT representation as well.

Modular Reduction. In our implementation, we need a total of six reduction module instantiations: While each BFU module contains a single reduction

Table 1: Resource Utilization on a XC7Z020 FPGA

Component	Parameter Set II	III	V
Look-Up Tables	24 320	29 987	42 860
Flip-Flops	9 668	11 274	14 136
Digital Signal Processors	45	45	45
Block-RAMs	15	23	33
f_{max}	140 MHz	142 MHz	127 MHz

module, the MACC module contains four reduction modules. For the modular reduction of a 46-bit value s, we recursively exploit the relation $2^{23} \equiv 2^{13} - 1 \bmod q$ in a similar way as in [16]. Our reduction utilizes four DSPs and two small LUT-based adders. For further details on the reduction, please refer to Appendix A.

Keccak. A fundamental part of Dilithium is the application of SHAKE-128 and SHAKE-256, both as hash function or as Extendable-Output Function (XOF). More precisely, both functions use the same Keccak permutation with the same state size of 1500 bits but a different rate r, which either is 1344 bits for SHAKE-128 or 1088 bits for SHAKE-256. Thus, our implementation features a single Keccak core that performs the permutation in 24 cycles (i.e., using a single cycle per round).

For data input and output we decided to implement 32-bit buses. During I/O operations, the Keccak module rotates the internal state for $r = 1344$ on a 32-bit basis while simultaneously, the input is added (exclusive-or) to the rotation feedback. Note that this behavior can also be used to compute SHAKE-256, i.e., by just using an unaltered feedback for the last $8 = (1344 - 1088)/32$ words.

Sampling. Dilithium requires several sampling algorithms that use the output of SHAKE. Unfortunately, none of the sampling algorithms is aligned to work on 32-bit words. We solved this problem using buffers with a length of the *least common multiple* of 32 and the desired output bit width. This enables converting a stream of 32-bit words to a stream of words with the desired output bit width.

Sampling the challenge c involves the Fisher-Yates shuffle. We implement this using a shift register with variable depth that contains all offsets of the non-zero coefficients and their sign bit. Once a random offset is found in rejection sampling, we *rotate* through the shift register and compare the stored offsets with the newly sampled one. If they are equal, we replace the old one with the current rejection threshold (keeping the sign bit), which essentially performs the swap. Then we increase the register depth and shift in the newly sampled offset with the corresponding sign bit. Finally, the polynomial is written to the BRAM.

For further details regarding making and checking the hints as well as rounding, refer to Appendix B and Appendix C.

Table 2: Performance Results

	Parameter Set								
	II			III			V		
	cycles		OP/s	cycles		OP/s	cycles		OP/s
Operation	[*min*]	[*avg*]	[*avg*]	[*min*]	[*avg*]	[*avg*]	[*min*]	[*avg*]	[*avg*]
KeyGen	18 600	18 761	7 462	32 943	33 102	4 290	50 669	50 982	2 491
Sign_{pre}	9 634	9 647	14 512	18 066	18 089	7 850	33 742	33 767	3 761
Sign	19 423	66 966	2 091	26 979	105 129	1 351	36 609	112 145	1 132
$\text{Verify}_{\text{pre}}$	10 890	10 917	12 824	19 945	19 966	7 112	36 244	36 250	3 503
Verify	8 759	8 770	15 963	12 072	12 084	11 751	16 447	16 462	7 715

4.2 Utilization and Performance Results

This section provides area utilization and performance results obtained after Place-and-Route (PnR) on a Xilinx XC7Z020 Artix-7 FPGA using the Vivado 2020.1 tool suite.

Utilization. Table 1 lists the results for resource utilization as well as the maximum frequency f_{max} obtained after synthesis and implementation. As expected, the LUT, FF, and BRAM utilization increases with the parameter sets, while the DSP utilization, governed by the NTT and MACC modules, is independent of the parameter sets.

Further, as the critical path delay mostly is determined by high net delays, reducing the area footprint, e.g., for dedicated sign-only or verify-only cores, can help to reduce delays and increase overall performance. For this, we would like to emphasize that even though we take advantage of sharing components for different operations, we still see potential for additional optimizations in this direction.

Performance. Table 2 shows performance results for our implementations, obtained after 1000 executions on random inputs.

For signature verification, we report cycle counts for valid signatures only. More precisely, since the norm check of $\mathbf{z}$, taking less than 100 cycles, is performed at the beginning, an invalid signature is processed subtantially faster. Besides, for signature generation, the cycle count spreads widely due to the nature of Dilithium. For this, we report both, minimum and average cycle counts. More specifically, the minimum number of cycles is close to the theoretical lower bound for the signature generation, corresponding to a best-case scenario in which a signature is accepted in the first iteration during simulation. However, for more realistic numbers, we also report average-case performance obtained after 1000 simulations on random inputs.

4.3 Comparison to Existing Work

The NTT uses 531 LUTs, 426 FFs, 17 DSPs and 1 BRAM in our core and takes 533/536 cycles for NTT/iNTT. The utilization of LUTs and FFs is smaller or

Table 3: Comparison of Hardware Design for PQC Signature Schemes

Oper.	Scheme	Platform	Utilization				f	t	Ref.
			LUT	FF	DSP	BRAM	MHz	μs	
KeyGen	Dilithium-III	XC7Z020	29987	11274	45	23	142	233	this
	Dilithium-III[2]	Virtex-7	54183	25236	182	15	350	52	[12]
	Dilithium-III[2,3]	Artix-7	86646	17674	–	–	119	1955	[15]
	qTesla-3[3]	Artix-7	111122	23398	–	–	79	45650	[15]
Sign	Dilithium-III	XC7Z020	29987	11274	45	23	142	740	this
	Dilithium-III[2]	Virtex-7	81530	83926	965	145	333	63	[12]
	Dilithium-III[2,3]	Artix-7	90567	21160	–	–	114	14140	[15]
	qTesla-3[3]	Artix-7	126008	25984	–	–	79	7441	[15]
	GLP	Spartan-6	7465	8993	28	29.5	–	1074	[10]
	Rainbow-Ia[4]	Kintex-7	27712	27679	0	59	111	18	[5]
	Rainbow-Ic[4]	Kintex-7	52895	32476	0	67	90	11	[5]
	SPHINCS-256	Kintex-7	19067	38132	3	36	525	1530	[1]
Verify	Dilithium-III	XC7Z020	29987	11274	45	23	142	85	this
	Dilithium-III[2]	Virtex-7	61738	34963	316	18	158	95	[12]
	Dilithium-III[2,3]	Artix-7	65274	15169	–	–	114	2491	[15]
	qTesla-3[3]	Artix-7	84834	17604	–	–	79	1926	[15]
	GLP	Spartan-6	6225	6663	8	15	–	1002	[10]

[2]round-2 parameters [3]High Level Synthesis [4]core enabling signing and verification

similar to existing implementations [13, 16] even though our modulus is larger. Also, since nearly all arithmetic is done in DSPs, our implementation results in low logic delays enabling high frequencies.

In Table 3, we compare our implementation of Dilithium-III with other relevant implementations of post-quantum signature schemes on reconfigurable hardware. While existing implementations of Dilithium for Artix-7 [15] and Virtex-7 [12] report area utilization, frequency, and latency individually per operation, we would like to emphasize that our core combines and embeds all operations in a single architecture.

Notably, our architecture outperforms existing solutions either in terms of resource utilization or throughput thus provides a compact, self-contained, and efficient solution for post-quantum secure digital signatures. In general, our design focuses on a reasonable trade-off between area consumption and performance degradation, in order to provide a modestly large and fast architecture.

5 Conclusion

In our paper, we present the first set of FPGA implementations for all three round-3 parameter sets of Dilithium for the low-end Artix-7 platform. Our design is a moderate proposal, featuring low latency compared to implementations of other post-quantum secure signature algorithms on the one hand, but still having a low area footprint on the other hand, making the usage of Dilithium feasible for many low-cost and constrained scenarios. As a highlight, our implementations are full-service processors for Dilithium, being capable of performing key generation, precomputations, signature generation, verification, arbitrary-length message digesting as well as key and signature packing and unpacking.

References

1. Dorian Amiet, Andreas Curiger, and Paul Zbinden. FPGA-based Accelerator for Post-Quantum Signature Scheme SPHINCS-256. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):18–39, 2018.
2. Utsav Banerjee, Tenzin S. Ukyab, and Anantha P. Chandrakasan. Sapphire: A Configurable Crypto-Processor for Post-Quantum Lattice-based Protocols. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(4):17–61, 2019.
3. Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):238–268, 2018.
4. Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium – Algorithm Specifications and Supporting Documentation (Version 3.1). Technical report, 02 2021. `https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf`.
5. Ahmed Ferozpuri and Kris Gaj. High-speed FPGA Implementation of the NIST Round 1 Rainbow Signature Scheme. In David Andrews, René Cumplido, Claudia Feregrino, and Dirk Stroobandt, editors, *2018 International Conference on ReConFigurable Computing and FPGAs, ReConFig 2018, Cancun, Mexico, December 3-5, 2018*, pages 1–8. IEEE, 2018.
6. Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gergor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-Fourier Lattice-basedCompact Signatures over NTRU - (Specification v1.2 - 01/10/2020). Technical report, October 2020. `https://falcon-sign.info/falcon.pdf`.
7. Denisa O. C. Greconici, Matthias J. Kannwischer, and Daan Sprenkels. Compact Dilithium Implementations on Cortex-M3 and Cortex-M4. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):1–24, 2021.
8. Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical Lattice-Based Cryptography: A Signature Scheme for Embedded Systems. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 530–547. Springer, 2012.
9. NIST. Call for Proposals - Post-Quantum Cryptography — CSRC. Technical report, NIST, 2017. `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization/Call-for-Proposals`.
10. Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. Enhanced Lattice-Based Signatures on Reconfigurable Hardware. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 353–370. Springer, 2014.
11. Post Quantum Cryptography Team. Post-Quantum Cryptography: NIST's Plan for the Future. Technical report, NIST, 2016. `https://csrc.nist.gov/csrc/media/projects/post-quantum-cryptography/documents/pqcrypto-2016-presentation.pdf`.

12. Sara Ricci, Lukas Malina, Petr Jedlicka, David Smekal, Jan Hajny, Petr Cibik, and Patrik Dobias. Implementing CRYSTALS-Dilithium Signature Scheme on FPGAs. Cryptology ePrint Archive, Report 2021/108, 2021. `https://eprint.iacr.org/2021/108`.
13. Sujoy Sinha Roy, Frederik Vercauteren, Nele Mentens, Donald Donglong Chen, and Ingrid Verbauwhede. Compact Ring-LWE Cryptoprocessor. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 371–391. Springer, 2014.
14. Peter W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134. IEEE Computer Society, 1994.
15. Deepraj Soni, Kanad Basu, Mohammed Nabeel, and Ramesh Karri. A hardware evaluation study of nist post-quantum cryptographic signature schemes. In *Second PQC Standardization Conference*. NIST, 2019.
16. Neng Zhang, Bohan Yang, Chen Chen, Shouyi Yin, Shaojun Wei, and Leibo Liu. Highly Efficient Architecture of NewHope-NIST on FPGA using Low-Complexity NTT/INTT. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):49–72, 2020.

A Reduction

We denote $s[a:b]$, where $a > b$, as the bit slice of s bounded by the offsets a, b counting from LSB to MSB, for example for $s = 6$ we have $s[2:1] = 11_2 = 3$.

$$
\begin{aligned}
&s[45:0] \equiv 2^{23}s[45:23] + s[22:0] \equiv 2^{13}s[45:23] - s[45:23] + s[22:0] \\
&\equiv 2^{23}s[45:33] + 2^{13}s[32:23] - s[45:23] + z \\
&\equiv 2^{13}\left(s[45:33] + s[45:23]\right) - \left(s[45:33] + s[45:23]\right) + z \\
&\equiv 2^{23}s[45:43] + 2^{13}\left(s[42:33] + s[32:23]\right) - \left(s[45:33] + s[45:23]\right) + z \\
&\equiv 2^{13}\left(s[45:43] + s[42:33] + s[32:23]\right) - \left(s[45:43] + s[45:33] + s[45:23]\right) + z \\
&\equiv 2^{13}x - y + z \equiv 2^{23}x[11:10] + 2^{13}x[9:0] - y + z \\
&\equiv 2^{13}\left(x[11:10] + x[9:0]\right) - \left(y + x[11:10]\right) + z \bmod q
\end{aligned}
$$

The result of our reduction can still be greater than 2^{23} so that we could repeat the substitution once again at the expense of additional depth and delay in the arithmetic computation. However, observing that the result of the reduction at this point is already within the interval $(-\mathbf{q}, 2\mathbf{q})$[3]. For this, we can simply add $\mathbf{q}$ to a negative result or subtract $\mathbf{q}$ if the result is positive. Eventually, delaying the reduced result, as well as given the sum and subtraction with $\mathbf{q}$, the final result is determined by selecting the non-negative value among the three computations.

In practice, we use four DSPs and two additions implemented in general-purpose logic to perform the modular reduction. The first DSP computes x and y by using a Kronecker substitution-like approach: The lower bits compute x and the higher bits compute y. However, as the computation does not fit entirely into the pre-adder stage, we need to add the least-significant bit of x using general-purpose logic outside the DSP and delay the result. Note, however, that for recent Ultrascale FPGAs, the witdh of pre-adder stage within the DSPs increased, which would allow to improve this reduction and give up the general-purpose addition.

Further, the second DSP computes $z + x[11:10] - y$ while in parallel, a LUT-based adder computes $x[11:10] + x[9:0]$, before both results are summed up in a third DSP. The fourth and final DSP adds $\mathbf{q}$ if the result of the third DSP is negative or subtracts $\mathbf{q}$ otherwise. Eventually, only the positive result is selected as output.

B Rounding

Implementing the Power2Round operation in hardware is very efficient, since during the computation of $\mathbf{t}$, we simply split result into the upper 10 bits and

[3]Since in our implementation all coefficients are stored in the standard representation $[0, q)$, this reduction also works for results of computations $ab + c$, since $(\mathbf{q} - 1)^2 + (\mathbf{q} - 1) < 2^{46}$.

the lower 13 bits, stored into different polynomial memories. However, since the $\mathbf{t_0}$ coefficients are interpreted as signed integers and our main paradigm is to store coefficients always as standard representatives, we need to add $\mathbf{q}$ if the most signification bit (MSB) is 1. Due to the structure of the operation, this is efficient with a LUT-based adder, which allows to avoid the additional usage of a DSP.

We implement the HighBits operation as a simple behavioral description of a range look-up depending on the input coefficient, which is efficient since for $\gamma_2 = (q-1)/32$, there are only 16 different possible output values while for $\gamma_2 = (q-1)/88$, there are only 44. Checking the low bits of $\mathbf{w} - c\mathbf{s_2}$, however, involves the MACC module in subtraction mode. Again, we implement a simple look-up that returns HighBits times $2\gamma_2$ and we subtract the result from the coefficient to obtain the low bits and check their norm without storing them.

C Hint

We store the hint in two registers, i.e., one storing the 1's offsets and the other one storing the k polynomial boundaries in the same format as specified for the packed signatures. For the MakeHint operation, we have $\mathbf{w} - c\mathbf{s_2}$ and $\mathbf{w} - c\mathbf{s_2} + c\mathbf{t_0}$ stored separately such that both can be read simultaneously. Eventually, we look up both HighBits and if differing, a new offset is shifted in. Further, for the UseHint operation, the hint module looks up the HighBits for each coefficient, i.e., both for h=0 and h=1. Then, selecting the correct one, the value is shifted into a buffer register for sampling (as described before) and absorbed to compute the value $\tilde{c}$, which ultimately is compared to the value of the signature during verification.